



Intersection-types *à la* Church

Luigi Liquori ^a, Simona Ronchi Della Rocca ^{b,*}

^a*INRIA Sophia Antipolis, France*

^b*Dipartimento di Informatica, Università di Torino, Italy*

Received 5 October 2005; revised 13 March 2007

Available online 2 June 2007

Abstract

In this paper, we present Λ_{\wedge}^t , a fully typed λ -calculus based on the intersection-type system discipline, which is a counterpart *à la* Church of the type assignment system as invented by Coppo and Dezani. The relationship between Λ_{\wedge}^t and the intersection type assignment system is the standard isomorphism between typed and type assignment system, and so the typed language inherits from the untyped system all the good properties, like subject reduction and strong normalization. Moreover, both type checking and type reconstruction are decidable.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Logics and intersection-types; λ -Calculus *à la* Curry and *à la* Church

1. Introduction

The implicative and conjunctive fragment of the intuitionistic logic (denoted by $\mathcal{L}_{\rightarrow}^{\wedge}$), where the logical connectives “ \rightarrow ” and “ \wedge ” denote the implication and the conjunction is a well-known powerful logical system: this logic is presented in Fig. 1.

The *Intersection-Type Assignment System* (\vdash_{\wedge}) is a set of inference rules for assigning *intersection-types* to terms of the untyped λ -calculus. Intersection-types are formulæ of the implicative and conjunctive fragment of propositional logic. The syntax and the typing rules are presented in Fig. 2. Intersection-types were introduced by Coppo and Dezani, to increase the typability power of Curry’s type assignment system for the λ -calculus [6]. Since then, intersection-types have been fruitfully used for designing static semantics of programming languages (*e.g.* Algol-like [17]), for characterizing interesting classes of λ -terms (*e.g.* the strongly normalizing ones [15]), and for studying denotational semantics of various untyped λ -calculi (*e.g.* [2,3]).

There are many versions in the literature of intersection-type assignment systems. Here we choose that one presented as “System D” [12], characterized by the presence of non syntax-directed rules for dealing with the introduction and elimination of the intersection. Note that, differently from most of the systems presented in the literature, as for example [7], in this system the connective \wedge is neither commutative nor associative nor idempotent. The choice of

* Corresponding author.

E-mail addresses: Luigi.Liquori@inria.fr (L. Liquori), ronchi@di.unito.it (S. Ronchi Della Rocca).

$$\begin{array}{c}
\text{Let } \Sigma \triangleq \{\sigma_1, \dots, \sigma_n\}, \text{ and } \Sigma, \sigma \triangleq \Sigma \cup \{\sigma\} \\
\\
\frac{\sigma \in \Sigma}{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma} (Var) \qquad \frac{\Sigma, \sigma_1 \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_2}{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1 \rightarrow \sigma_2} (\rightarrow I) \\
\\
\frac{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1 \rightarrow \sigma_2 \quad \Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1}{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_2} (\rightarrow E) \qquad \frac{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1 \quad \Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_2}{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1 \wedge \sigma_2} (\wedge I) \\
\\
\frac{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1 \wedge \sigma_2}{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1} (\wedge E_L) \qquad \frac{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_1 \wedge \sigma_2}{\Sigma \vdash_{\mathcal{L}_{\rightarrow}^{\wedge}} \sigma_2} (\wedge E_R)
\end{array}$$

Fig. 1. The logic $\mathcal{L}_{\rightarrow}^{\wedge}$.**Syntax of Λ_{\wedge}^u**

Let α range over a denumerable set \mathcal{V} of type-constants

Let “ \wedge ” take precedence over “ \rightarrow ”

$$M ::= x \mid \lambda x. M \mid M M$$

$$\sigma ::= \alpha \mid \sigma \rightarrow \sigma \mid \sigma \wedge \sigma$$

Type System for Λ_{\wedge}^u

Let $E \triangleq \{x_1:\sigma_1, \dots, x_n:\sigma_n\}$ ($i \neq j$ implies $x_i \neq x_j$), and $E, x:\sigma \triangleq E \cup \{x:\sigma\}$

$$\begin{array}{c}
\frac{x:\sigma \in E}{E \vdash_{\wedge} x : \sigma} (Var) \qquad \frac{E, x:\sigma_1 \vdash_{\wedge} M : \sigma_2}{E \vdash_{\wedge} \lambda x. M : \sigma_1 \rightarrow \sigma_2} (\rightarrow I) \\
\\
\frac{E \vdash_{\wedge} M : \sigma_1 \rightarrow \sigma_2 \quad E \vdash_{\wedge} N : \sigma_1}{E \vdash_{\wedge} M N : \sigma_2} (\rightarrow E) \qquad \frac{E \vdash_{\wedge} M : \sigma_1 \quad E \vdash_{\wedge} M : \sigma_2}{E \vdash_{\wedge} M : \sigma_1 \wedge \sigma_2} (\wedge I) \\
\\
\frac{E \vdash_{\wedge} M : \sigma_1 \wedge \sigma_2}{E \vdash_{\wedge} M : \sigma_1} (\wedge E_L) \qquad \frac{E \vdash_{\wedge} M : \sigma_1 \wedge \sigma_2}{E \vdash_{\wedge} M : \sigma_2} (\wedge E_R)
\end{array}$$

Fig. 2. The Intersection-Type Assignment System \vdash_{\wedge} .

this presentation has been taken since we are looking for a typed version of the calculus, where bound variables come decorated with their types, and in this setting it is natural to consider types as syntactical entities. In any case, this presentation does not have any consequence on the typability power of the intersection type assignment system, which is well known to characterize all and only the strongly normalizing terms [12, 15].

Following the standard terminology, let us call *à la* Curry a system assigning types to untyped terms, and *à la* Church a system assigning types to typed terms, *i.e.* where types are part of the syntax of terms, by decorating bound-variables in abstractions. Differently from other type assignment systems *à la* Curry, Λ_{\wedge}^u has no natural counterpart *à la* Church. The classical example is the polymorphic identity in Λ_{\wedge}^u that has the following type-derivation:

$$\frac{\frac{x:\sigma_1 \vdash_{\wedge} x : \sigma_1}{\vdash_{\wedge} \lambda x. x : \sigma_1 \rightarrow \sigma_1} (\rightarrow I) \quad \frac{x:\sigma_2 \vdash_{\wedge} x : \sigma_2}{\vdash_{\wedge} \lambda x. x : \sigma_2 \rightarrow \sigma_2} (\rightarrow I)}{\vdash_{\wedge} \lambda x. x : (\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)} (\wedge I)$$

but is untypable using a naïve corresponding rule *à la* Church for the introduction of intersection-types [10].

$$\frac{\frac{x:\sigma_1 \vdash_{\wedge} x : \sigma_1}{\vdash_{\wedge} \lambda x. \sigma_1. x : \sigma_1 \rightarrow \sigma_1} (\rightarrow I) \quad \frac{x:\sigma_2 \vdash_{\wedge} x : \sigma_2}{\vdash_{\wedge} \lambda x. \sigma_2. x : \sigma_2 \rightarrow \sigma_2} (\rightarrow I)}{\vdash_{\wedge} \lambda x. [\Box]. x : (\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)} (\wedge I)$$

By the Curry-Howard isomorphism [11], a λ -term must record the shape of its type-derivation. A standard proof decoration would give rise to a language which is a λ -calculus extended with a pair construction. For example, according to [18], the previous proof would be decorated in the following way:

$$\frac{\frac{x:\sigma_1 \vdash_{\wedge} x:\sigma_1}{\vdash_{\wedge} \lambda x:\sigma_1.x:\sigma_1 \rightarrow \sigma_1} (\rightarrow I) \quad \frac{x:\sigma_2 \vdash_{\wedge} x:\sigma_2}{\vdash_{\wedge} \lambda x:\sigma_2.x:\sigma_2 \rightarrow \sigma_2} (\rightarrow I)}{\vdash_{\wedge} \langle \lambda x:\sigma_1.x, \lambda x:\sigma_2.x \rangle : (\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)} (\wedge I)$$

The resulting language has a huge syntax, since the pairing construct can be applied only on terms which can be different, but their below untyped versions must be identical. An example in the literature of λ -calculus typed *à la* Church with intersection types, where the syntax is exactly the classical one, but for types, is the language Forsythe in [17]. But it is incomplete, in the sense that the resulting typed system has less typability power than the type assignment one. In fact, in the Reynolds syntax, assuming that a term M has type τ under the assumption that the variable x has any one of the types σ_i ($1 \leq i \leq n$), we can form the typed term:

$$\lambda x:\sigma_1|\sigma_2|\dots|\sigma_n.M$$

having types $(\sigma_i \rightarrow \tau)$ for $1 \leq i \leq n$, and all types derived from these by applying intersection introduction, intersection elimination and subtyping relations. So, for example, there is not a typed version of $\lambda x.\lambda y.x$, giving it the type $\rho = (\sigma \rightarrow (\sigma \rightarrow \sigma)) \wedge (\tau \rightarrow (\tau \rightarrow \tau))$, where σ and τ are uncomparable. In fact, according to the Forsythe syntax, we can form the two terms, namely $\lambda x:\sigma|\tau.\lambda y:\sigma.x$, having types $\sigma \rightarrow (\sigma \rightarrow \sigma)$ and $\tau \rightarrow (\sigma \rightarrow \tau)$, and $\lambda x:\sigma|\tau.\lambda y:\tau.x$, having types $\tau \rightarrow (\tau \rightarrow \tau)$ and $\sigma \rightarrow (\tau \rightarrow \sigma)$, but there is not a term relating the types of x and y in the desired way. (this example has been taken from [21]). Moreover, Forsythe appear something in between a typed and a type assignment calculus, since a term does not have a unique type.

The problem is, as the skilled reader can understand, the presence of *non syntax-directed rules* that disconnect the λ -term from its type-derivation (hence losing the Curry-Howard correspondence). It is important to point out that this problem does not depend on the chosen intersection-type assignment system; indeed, not one of the intersection type assignment systems presented in the literature is completely syntax directed (and not-even it cannot be!).

Our goal is to build a λ -calculus *à la* Church, and related intersection-type system Λ_{\wedge}^t , whose syntax is, as far as possible, similar to other typed λ -calculi. We want to design this calculus through a typed system, building typed terms together with their type, such that the typed system and the type assignment system Λ_{\wedge}^u are related by the standard path designed in [8, 14, 4]. For this system, we are interested to catch as much as possible properties from the following list:

Desiderata

- (1) typed and type assignment derivations are *isomorphic*, under the assumption that they share the same type syntax. That is, the application of an *erasing function* \mathcal{E} on all typed terms and contexts (in a typed derivation judgment) produces a derivable type assignment derivation with the same structure, and every type assignment derivation is obtained from a typed one with the same structure by applying the same erasure \mathcal{E} . Such a kind of isomorphism has been studied in [4].

Moreover, we want that the intersection calculus *à la* Church inherits all the properties of intersection type assignment *à la* Curry, namely:

- (2) subject reduction;
 - (3) strong normalization of typable terms;
- plus the following ones:
- (4) unicity of typing;
 - (5) decidability of type reconstruction and of type checking.

In order to find a solution to this challenge, we designed a calculus, whose terms are composed by two parts, carrying out the computational and logical information, respectively. The first component (the *marked-term*) is a simply typed λ -term, but types are variable-marks, *i.e.*, natural numbers representing store locations. The second component (the *proof-term*) records both the associations between variable-marks and types and the structure of the

derivation. The technical tool for realizing this is an unusual formulation of context, assigning types to term-variables *at a given mark/location*. The calculus of proof-terms can be defined *per se*, as decoration of the implicative and conjunctive fragment of intuitionistic logic; it codifies a set of proofs that is strictly bigger than these corresponding to intersection-type derivations (see [19]).

As example, the typed identity with type

$$(\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)$$

can be written in our proposal as the term

$$(\lambda x:0.x)@(\lambda 0:\sigma_1.0) \wedge (\lambda 0:\sigma_2.0)$$

where 0 is a mark, and $(\lambda 0:\sigma_1.0) \wedge (\lambda 0:\sigma_2.0)$ is the logical content of $\lambda x:0.x$. The typed λ -calculus so obtained satisfies all the above requirements. As a nice consequence of these choices, we get decidability of the type reconstruction and type checking, both being also easy to define.

There are other proposals in the literature for a λ -calculus typed with intersection types. We already have cited Forsythe [17], which is not complete, as well as the language proposed in [16]. The languages proposed in [5,20,21] have been designed with other purposes, and they do not satisfy requirement 1. The language in [18] has been designed for logical purposes in order to satisfy the requirement 1, but its syntax and operational semantics are unsatisfying, from our point of view.

The paper is organized as follows: Section 2 presents the logical calculus. Section 3 shows the whole intersection-typed λ -calculus. Section 4 contains some examples, while Section 5 lists the metatheory and the type checking/type reconstruction algorithms. In Section 6 the isomorphism between Λ_{\wedge}^t and Λ_{\wedge}^u is proved. Conclusions and final remarks end the paper.

2. The Proof-calculus $\Lambda\mathcal{P}$

The syntax of intersection-types is that of the formulas of the implicative and conjunctive fragment of the intuitionistic logic, where the logical connectives “ \rightarrow ” and “ \wedge ” denote the implication and the conjunction. Unfortunately, the intersection-type assignment system Λ_{\wedge}^u does not correspond, in the Curry–Howard sense, to this logic [10], because of the “anomalous” decoration of the rules dealing with conjunction.

In what follows, we present a typed λ -calculus, obtained by decorating the proof of such a logic. The main peculiarity of this calculus is that it is defined on another categories of variables called *variable-marks*; the calculus will be used to record the structure of an intersection derivation, though an association between variable-marks and types.

Syntax of $\Lambda\mathcal{P}$. We start with some useful definitions.

Definition 2.1.

- (1) Variable-marks (denoted by ι) range over \mathbf{Nat} ;
- (2) Intersection-types are defined as follows:

$$\sigma ::= \alpha \mid \sigma \rightarrow \sigma \mid \sigma \wedge \sigma$$

where α ranges over a denumerable set \mathcal{V} of constants;

- (3) Proof-contexts are finite associations between variable-marks and types, and they are defined by the following grammar:

$$G ::= \epsilon \mid G, \iota:\sigma$$

- (4) Pseudo-proof-trees are labeled unary/binary trees defined as follows:

$$\Delta ::= \iota \mid \lambda \iota:\sigma. \Delta \mid \Delta \Delta \mid \Delta \wedge \Delta \mid \Delta \rightarrow \Delta \mid \Delta \searrow$$

The Proof-calculus $\Lambda\mathcal{P}$.

Let $G \triangleq \{\iota_1:\sigma_1, \dots, \iota_n:\sigma_n\}$ ($i \neq j$ implies $\iota_i \not\equiv \iota_j$), and $G, \iota:\sigma \triangleq G \cup \{\iota:\sigma\}$

$$\begin{array}{c}
 \frac{\iota:\sigma \in G}{G \vdash_P \iota : \sigma} (Var) \qquad \frac{G, \iota:\sigma_1 \vdash_P \Delta : \sigma_2}{G \vdash_P \lambda\iota:\sigma_1. \Delta : \sigma_1 \rightarrow \sigma_2} (\rightarrow I) \\
 \\
 \frac{G \vdash_P \Delta_1 : \sigma_1 \rightarrow \sigma_2 \quad G \vdash_P \Delta_2 : \sigma_1}{G \vdash_P \Delta_1 \Delta_2 : \sigma_2} (\rightarrow E) \qquad \frac{G \vdash_P \Delta_1 : \sigma_1 \quad G \vdash_P \Delta_2 : \sigma_2}{G \vdash_P \Delta_1 \wedge \Delta_2 : \sigma_1 \wedge \sigma_2} (\wedge I) \\
 \\
 \frac{G \vdash_P \Delta : \sigma_1 \wedge \sigma_2}{G \vdash_P \Delta \searrow : \sigma_1} (\wedge E_L) \qquad \frac{G \vdash_P \Delta : \sigma_1 \wedge \sigma_2}{G \vdash_P \Delta \swarrow : \sigma_2} (\wedge E_R)
 \end{array}$$

Fig. 3. The Proof-calculus $\Lambda\mathcal{P}$.

(5) The set $Fv(\Delta)$ of the variable-marks in a pseudo-proof-tree Δ is

$$\begin{array}{ll}
 Fv(\iota) \triangleq \{\iota\} & Fv(\lambda\iota:\sigma.\Delta) \triangleq Fv(\Delta) \setminus \{\iota\} \\
 Fv(\Delta \searrow) \triangleq Fv(\Delta) & Fv(\Delta_1 \Delta_2) \triangleq Fv(\Delta_1) \cup Fv(\Delta_2) \\
 Fv(\Delta \swarrow) \triangleq Fv(\Delta) & Fv(\Delta_1 \wedge \Delta_2) \triangleq Fv(\Delta_1) \cup Fv(\Delta_2)
 \end{array}$$

A variable-mark is bound in Δ if it is not free in Δ .

Type system for $\Lambda\mathcal{P}$. The system proves judgments of the shape:

$$G \vdash_P \Delta : \sigma$$

where G is a proof-context, Δ is a pseudo-proof-tree, and σ is a type. The pseudo-proof-tree Δ is a *legal* proof-tree if there are G and σ such that $G \vdash_P \Delta : \sigma$. The rules of the system, obtained by decorating the rules of the logic $\mathcal{L}_{\rightarrow}^{\wedge}$, are showed in Fig. 3. Note that $\Lambda\mathcal{P}$ is just an unusual syntax for the simply typed λ -calculus with pairs, which can be seen, via the Curry-Howard isomorphism, as a decoration of $\mathcal{L}_{\rightarrow}^{\wedge}$.

Reduction semantics of $\Lambda\mathcal{P}$. Being $\Lambda\mathcal{P}$ a calculus isomorphic to the typed λ -calculus with pairs, its reduction rules are the well known ones:

$$\begin{array}{l}
 (\lambda\iota:\sigma.\Delta_1) \Delta_2 \rightarrow_{\beta} \Delta_1[\Delta_2/\iota] \\
 \Delta \searrow (\Delta_1 \wedge \Delta_2) \rightarrow_{\pi_1} \Delta_1 \\
 (\Delta_1 \wedge \Delta_2) \swarrow \rightarrow_{\pi_2} \Delta_2
 \end{array}$$

As usual, the $\Lambda\mathcal{P}$ calculus works modulo α -conversion, as the symmetric, transitive, reflexive, and contextual closure of the following rule:

$$\lambda\iota_1:\sigma.\Delta \rightarrow_{\alpha} \lambda\iota_2:\sigma.\Delta[\iota_2/\iota_1] \quad \text{where } \iota_2 \text{ is fresh}$$

The following result holds:

Fact 1 (Strong normalization of $\Lambda\mathcal{P}$). $\Lambda\mathcal{P}$ is strongly normalizing.

3. The intersection-typed calculus Λ_{\wedge}^t

The key idea in the design of the intersection-typed system is to split the term into two parts, carrying out the computational and the logical information, respectively. Namely, the first one is a term of a typed λ -calculus, while the second one is a proof-term (belonging to the language $\Lambda\mathcal{P}$ introduced in the previous section) describing the shape of the type derivation. The technical tool for connecting the two parts is an unusual formulation of contexts. In fact, a context associates to a variable both a variable-mark *and* a type, such that different variables are associated to different variable-marks.

This novel formulation of contexts allows to remember, in rule $(\rightarrow I)$, just the variable-mark, the corresponding type being stored in the proof-tree, built by the system *in parallel* with the typed term. In this way the underlying term is *de facto* a term of the classical untyped λ -calculus. Since the proof-tree describes the structure of the type-derivation, we also obtain the decidability of type reconstruction and type checking.

Syntax

Definition 3.1.

- (1) Contexts are finite associations between variables and types at a given variable-mark, such that each variable and each variable-mark occur at least once in it. They are defined as follows:

$$\Gamma ::= \epsilon \mid \Gamma, x@l:\sigma$$

- (2) Marked-terms are defined as follows:

$$M ::= x \mid \lambda x:l.M \mid MM$$

- (3) A pseudo-term of Λ_{\wedge}^t has the shape $M@\Delta$, where M is a marked-term and Δ is a proof-tree;

In what follows, the symbol \equiv denotes the syntactic equality for marked-terms, types, contexts, variable-marks and proof-trees, respectively.

Definition 3.2 (Fv).

- (1) The set of free-variables of a marked-term is defined as follows:

$$\text{Fv}(x) \triangleq \{x\} \quad \text{Fv}((\lambda x:l.M)) \triangleq \text{Fv}(M) \setminus \{x\} \quad \text{Fv}(MN) \triangleq \text{Fv}(M) \cup \text{Fv}(N)$$

- (2) The set of free variables and of free variable-marks of a pseudo term is defined as follows:

$$\text{Fv}(M@\Delta) \triangleq \text{Fv}(M) \cup \text{Fv}(\Delta)$$

Type system \vdash . The type judgments of the intersection-typed calculus Λ_{\wedge}^t have the shape:

$$\Gamma \vdash M@\Delta : \sigma$$

where Γ is a context, M is a marked-term, and Δ is a proof-tree. Intuitively: in the judgment, the type-context Γ assigns intersection-types to the free-variables of M annotated by variable-marks; if $\Gamma \vdash M@\Delta : \sigma$, then we say that $M@\Delta$ is a term of Λ_{\wedge}^t .

The proof-tree keeps track of the type of the used mark together with a trace of the *skeleton* of the derivation tree. The proof-tree Δ plays the role of a road map to backtrack (*i.e.* roll back) the derivation tree. The typing rules are presented in Fig. 4. Some comments are in order:

$$\begin{array}{c}
\frac{x@l:\sigma \in \Gamma}{\Gamma \vdash x@l : \sigma} (Var) \qquad \frac{\Gamma, x@l:\sigma_1 \vdash M@ \Delta : \sigma_2}{\Gamma \vdash (\lambda x:l.M)@(\lambda l:\sigma_1.\Delta) : \sigma_1 \rightarrow \sigma_2} (\rightarrow I) \\
\\
\frac{\Gamma \vdash M@ \Delta_1 : \sigma_1 \rightarrow \sigma_2 \quad \Gamma \vdash N@ \Delta_2 : \sigma_1}{\Gamma \vdash (M N)@(\Delta_1 \Delta_2) : \sigma_2} (\rightarrow E) \qquad \frac{\Gamma \vdash M@ \Delta_1 : \sigma_1 \quad \Gamma \vdash M@ \Delta_2 : \sigma_2}{\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2} (\wedge I) \\
\\
\frac{\Gamma \vdash M@ \Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\swarrow \Delta) : \sigma_1} (\wedge E_L) \qquad \frac{\Gamma \vdash M@ \Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\searrow \Delta) : \sigma_2} (\wedge E_R)
\end{array}$$

Fig. 4. The type system \vdash for Λ_{\wedge}^t .

- (*Var*) gives types to free-variables at a given mark;
- ($\rightarrow I$) is a quasi-classical abstraction rule, but it records in the term only the variable-mark associated to the abstracted variable; the proof-tree Δ evolves in a new proof-tree enriched with the binding for the mark l ;
- ($\rightarrow E$) is a quasi-classical application rule; observe that the two proof-trees of the premises become sub proof-trees in the conclusion (the hidden application operator being the root);
- ($\wedge I$) is the most important rule; given two judgments for M assigning types σ_1 and type σ_2 , in the same context Γ but with different proof-trees Δ_1 , and Δ_2 , we can assign the intersection-type $\sigma_1 \wedge \sigma_2$ to M in the context Γ but in the new proof-tree $\Delta_1 \wedge \Delta_2$. At this point the marked-term M *loses the one-to-one correspondence with its proof*. Luckily, the new proof-tree keeps track of the derivation and guarantees unicity of typing;
- ($\wedge E_L$), v and ($\wedge E_R$) are the two standard rules that eliminate intersection-types. Also in this case the marked-term M *loses the one-to-one correspondence with its (logical) proof*, but the proof is memorized by the proof-tree, thanks to the two place-holders \swarrow and \searrow , indicating the applied rule.

Reduction semantics. For a given term $M@ \Delta$, the computational part (M) and the logical part (Δ) grow up together while they are built through application of rules (*Var*), ($\rightarrow I$), and ($\rightarrow E$), but they *get disconnected* when we apply ($\wedge I, E_L, E_R$) rules (that changes the Δ but not the M). This disconnection is “logged” in the Δ via occurrences of operators \wedge , \swarrow , and \searrow .

As such:

- by ($\wedge I$) and from $M@ \Delta_1$ and $M@ \Delta_2$, we get $M@(\Delta_1 \wedge \Delta_2)$. So Δ_1 and Δ_2 describe two different derivations sharing the same marked term. As a consequence, to a redex in M will correspond some redexes in Δ_1 and Δ_2 , that need to be performed in parallel with the redex in M in order to preserve the correct syntax of the term;
- by ($\wedge E_L$) or ($\wedge E_R$) and from $M@ \Delta$, we get $M@(\swarrow \Delta)$ or $M@(\searrow \Delta)$; so each reduction in M implies some reductions inside the wrapped Δ .

In order to correctly identify the reductions that need to be performed in parallel in the two parts of the term, we will define the notion of overlapping. Namely a redex is defining taking into account the surrounding context.

Definition 3.3 (*Reduction semantics*).

(Contexts and Overlapping) The marked-term contexts $C\{\}$ and proof-term context $D\{\}$ and multi-hole proof-term context $E\{\}$ are defined as follows:

$$\begin{aligned}
C\{\} &::= \{\} \mid C\{\} M \mid M C\{\} \mid \lambda x:l.C\{\} \\
D\{\} &::= \{\} \mid D\{\} \Delta \mid \Delta D\{\} \mid \lambda l:\sigma.D\{\} \mid \swarrow D\{\} \mid D\{\} \searrow \mid D\{\} \wedge \Delta \mid \Delta \wedge D\{\} \\
E\{\} &::= \{\} \mid E\{\} \Delta \mid \Delta E\{\} \mid \lambda l:\sigma.E\{\} \mid \swarrow E\{\} \mid E\{\} \searrow \mid E\{\} \wedge E\{\}
\end{aligned}$$

Note that, while $C\{\}$ and $D\{\}$ are contexts with exactly one hole, the context $E\{\}$ can have more than one hole. Let $E\{\Delta_i\}_{i \in I}$ denote a proof-context where the i -th hole has been filled by Δ_i , for $i \in I$. The notion of overlapping between marked-terms and proof-terms, denoted by $C\{\} \sqsubseteq E\{\}$ is defined by induction on $E\{\}$ as follows:

$$\frac{}{\{\} \sqsubseteq \{\}} \quad \frac{C\{\} \sqsubseteq E\{\}}{C\{\}M \sqsubseteq E\{\}\Delta} \quad \frac{C\{\} \sqsubseteq E\{\}}{M C\{\} \sqsubseteq \Delta E\{\}} \quad \frac{C\{\} \sqsubseteq E\{\}}{\lambda x:\iota. C\{\} \sqsubseteq \lambda \iota:\sigma. E\{\}}$$

$$\frac{C\{\} \sqsubseteq E\{\}}{C\{\} \sqsubseteq \swarrow E\{\}} \quad \frac{C\{\} \sqsubseteq E\{\}}{C\{\} \sqsubseteq E\{\}\searrow} \quad \frac{C\{\} \sqsubseteq E_1\{\} \quad C\{\} \sqsubseteq E_2\{\}}{C\{\} \sqsubseteq E_1\{\} \wedge E_2\{\}}$$

(\rightarrow) **reduction** The (\rightarrow) reduction rule is the union of the three reduction rules defines as follows:

$$\text{if } C\{\} \sqsubseteq E\{\} \text{ then}$$

$$C\{(\lambda x:\iota. M)N\}@E\{(\lambda \iota:\sigma_i. \Delta_i) \Delta'_i\}_{i \in I} \rightarrow_\beta C\{M[N/x]\}@E\{\Delta_i[\Delta'_i/\iota]\}_{i \in I}$$

$$M@D\{\swarrow(\Delta_1 \wedge \Delta_2)\} \rightarrow_{\pi_1} M@D\{\Delta_1\}$$

$$M@D\{(\Delta_1 \wedge \Delta_2)\searrow\} \rightarrow_{\pi_2} M@D\{\Delta_2\}$$

Note that the hypothesis of $C\{\} \sqsubseteq E\{\}$ in the definition of the \rightarrow_β -reduction is essential in order to recover, for each β -redex in the marked term, the corresponding β -redexes, wrapped in the proof-term; those redexes must fired in parallel, in order to do not create exotic typed terms that would not have an untyped counterpart in the original system *à la* Curry.

α -conversion on well formed terms can be formally defined as follows:

Definition 3.4 (α -Conversion).

$$(\lambda x:\iota. M)@\Delta \rightarrow_\alpha (\lambda y:\iota. M[y/x])@\Delta \quad y \text{ fresh in } M$$

$$M@(\lambda \iota_1:\sigma. \Delta) \rightarrow_\alpha M[t_2/t_1]@(\lambda \iota_2:\sigma. \Delta[t_2/t_1]) \quad t_2 \text{ fresh in } \Delta$$

4. Examples

We show two notorious examples in order to justify how type derivations can be built for Λ_\wedge^t proof-terms starting for the corresponding untyped λ -terms *à la* Curry, and two further examples in order to illustrate the reduction rules of Λ_\wedge^t .

Example 4.1 (Classical polymorphic identity).

We show a derivation for a typed term corresponding to the typing in $\Lambda_\wedge^u : \vdash_\wedge \lambda x.x : (\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)$.

$$\frac{\frac{x@0:\sigma_1 \vdash x@0:\sigma_1}{\vdash (\lambda x:0.x)@(\lambda 0:\sigma_1.0) : \sigma_1 \rightarrow \sigma_1} (Var) \quad \frac{x@0:\sigma_2 \vdash x@0:\sigma_2}{\vdash (\lambda x:0.x)@(\lambda 0:\sigma_2.0) : \sigma_2 \rightarrow \sigma_2} (Var)}{\vdash (\lambda x:0.x)@((\lambda 0:\sigma_1.0) \wedge (\lambda 0:\sigma_2.0)) : (\sigma_1 \rightarrow \sigma_1) \wedge (\sigma_2 \rightarrow \sigma_2)} (\wedge I)$$

Example 4.2 (Polymorphic self-application).

Let $\sigma_2 \triangleq (\sigma_1 \rightarrow \sigma_1) \wedge \sigma_1$. We show a term in Λ_\wedge^t corresponding to the typing of $\Lambda_\wedge^u : \vdash_\wedge \lambda x.xx : \sigma_2 \rightarrow \sigma_1$.

$$\frac{\frac{x@0:\sigma_2 \vdash x@0:\sigma_2}{x@0:\sigma_2 \vdash x@(\swarrow 0) : \sigma_1 \rightarrow \sigma_1} (Var) \quad \frac{x:0:\sigma_2 \vdash x@0:\sigma_2}{x:0:\sigma_2 \vdash x@(\searrow 0) : \sigma_1} (Var)}{\frac{x@0:\sigma_2 \vdash x@(\swarrow 0) (\searrow 0) : \sigma_1}{\vdash (\lambda x:0.x)@(\lambda 0:\sigma_2.(\swarrow 0) (\searrow 0)) : \sigma_2 \rightarrow \sigma_1} (\rightarrow I)} (\wedge E_L) \quad (\wedge E_R) \quad (\rightarrow E)$$

Note how the proof-tree memorizes exactly the skeleton of the type-derivation.

Example 4.3 (Reduction in Λ_{\wedge}^t).

A term in Λ_{\wedge}^t corresponding to the following typing in $\Lambda_{\wedge}^u : y:\sigma_1 \wedge \sigma_2 \vdash_{\wedge} (\lambda x.x) y : \sigma_1 \wedge \sigma_2$ can be constructed in the following way:

$$\frac{\mathcal{D}_1 \quad \mathcal{D}_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x) y@((\lambda l:\sigma_1.l) \swarrow 0) \wedge ((\lambda l:\sigma_2.l) 0_{\searrow}) : \sigma_1 \wedge \sigma_2} (\wedge I)$$

where \mathcal{D}_1 is:

$$\frac{\frac{y@0:\sigma_1 \wedge \sigma_2, x@1:\sigma_1 \vdash x@1 : \sigma_1}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x)@(\lambda l:\sigma_1.l) : \sigma_1 \rightarrow \sigma_1} (Var) \quad \frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (Var)}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x)@(\lambda l:\sigma_1.l) : \sigma_1 \rightarrow \sigma_1} (\rightarrow I) \quad \frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (\wedge E_L)}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x) y@((\lambda l:\sigma_1.l) \swarrow 0) : \sigma_1} (\rightarrow E)$$

and \mathcal{D}_2 is:

$$\frac{\frac{y@0:\sigma_1 \wedge \sigma_2, x@1:\sigma_2 \vdash x@1 : \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x)@(\lambda l:\sigma_2.l) : \sigma_2 \rightarrow \sigma_2} (Var) \quad \frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (Var)}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x)@(\lambda l:\sigma_2.l) : \sigma_2 \rightarrow \sigma_2} (\rightarrow I) \quad \frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (\wedge E_R)}{y@0:\sigma_1 \wedge \sigma_2 \vdash (\lambda x:l.x) y@((\lambda l:\sigma_2.l) 0_{\searrow}) : \sigma_2} (\rightarrow E)$$

This term can be written as: $\mathbf{C}\{(\lambda x:l.x) y\}@\mathbf{E}\{(\lambda l:\sigma_1.l) \swarrow 0\}\{(\lambda l:\sigma_2.l) 0_{\searrow}\}$, where the two overlapping contexts are respectively, $\mathbf{C}\{\} \equiv \{\}$ and $\mathbf{E}\{\} \equiv \{\} \wedge \{\}$, so this term β -reduces to: $y@0 \wedge 0_{\searrow}$ which is a well formed term, in fact it can be built in the following way:

$$\frac{\frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (Var) \quad \frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (Var)}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (\wedge E_L) \quad \frac{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (\wedge E_R)}{y@0:\sigma_1 \wedge \sigma_2 \vdash y@0:\sigma_1 \wedge \sigma_2} (\wedge I)$$

Note that, if only one of the redexes in the proof-term would be reduced, then the resulting term will be not well formed.

Example 4.4 (A complete reduction).

Consider the typing in Λ_{\wedge}^u

$$\vdash_{\wedge} (\lambda x.x x) ((\lambda y.y) (\lambda y.y)) : \sigma \wedge \tau$$

where $\sigma \triangleq \alpha \rightarrow \alpha$ and $\tau \triangleq \beta \rightarrow \beta$. A corresponding term in Λ_{\wedge}^t is:

$$\vdash (\lambda x:0.x x) ((\lambda y:1.y) (\lambda y:2.y))@(\Delta_1 \Delta_2) : \sigma \wedge \tau$$

where

$$\delta \triangleq ((\sigma \rightarrow \sigma) \wedge \sigma) \wedge ((\tau \rightarrow \tau) \wedge \tau)$$

and

$\Delta_1 = \lambda 0 : \delta. \Delta_3$	$\Delta_3 = \Delta_4 \wedge \Delta_5$	$\Delta_4 = \Delta_6 \Delta_7$	$\Delta_5 = \Delta'_6 \Delta'_7$
$\Delta_6 = \swarrow (\swarrow 0)$	$\Delta_7 = (\swarrow 0)_{\searrow}$	$\Delta'_6 = \swarrow (0_{\searrow})$	$\Delta'_7 = (0_{\searrow})_{\searrow}$
$\Delta_2 = \Delta_8 \wedge \Delta_9$	$\Delta_8 = \Delta_{10} \wedge \Delta_{11}$	$\Delta_{10} = \Delta_{12} \Delta_{13}$	$\Delta_{12} = \lambda 1:\sigma^2.2$
$\Delta_{13} = \lambda 2:\sigma.2$	$\Delta_{11} = \Delta_{14} \Delta_{15}$	$\Delta_{14} = \lambda 1:\sigma.1$	$\Delta_{15} = \lambda 2:\alpha.2$
$\Delta_9 = \Delta'_{10} \wedge \Delta'_{11}$	$\Delta'_{10} = \Delta'_{12} \Delta'_{13}$	$\Delta'_{12} = \lambda 1:\tau^2.1$	$\Delta'_{13} = \lambda 2:\tau.2$
$\Delta'_{11} = \Delta'_{14} \Delta'_{15}$	$\Delta'_{14} = \lambda 1:\tau.1$	$\Delta'_{15} = \lambda 2:\beta.1$	

Reducing the top-level β -redex (corresponding to the two overlapping contexts $\mathbf{C}\{\} \equiv \mathbf{E}\{\} \equiv \{\}$), leads to the term

$$((\lambda y:1.y) (\lambda y:2.y)) ((\lambda y:1.y) (\lambda y:2.y)) @ (\underbrace{(\swarrow(\swarrow\Delta_2))}_{\pi\text{-red}}) (\underbrace{(\swarrow\Delta_2)\searrow)}_{\pi\text{-red}}) \wedge (\underbrace{(\swarrow(\Delta_2)\searrow)}_{\pi\text{-red}}) (\underbrace{(\Delta_2)\searrow\searrow)}_{\pi\text{-red}})$$

Reducing all the four π -redexes leads to the following term

$$((\lambda y:1.y) (\lambda y:2.y)) ((\lambda y:1.y) (\lambda y:2.y)) @ (\Delta_{10} \Delta_{11}) \wedge (\Delta'_{10} \Delta'_{11})$$

that can be seen both as:

$$\mathbf{C}\{(\lambda y:1.y) (\lambda y:2.y)\} @ \mathbf{E}\{\Delta_{10}\}\{\Delta'_{10}\}$$

and

$$\mathbf{C}\{(\lambda y:1.y) (\lambda y:2.y)\} @ \mathbf{E}\{\Delta_{11}\}\{\Delta'_{11}\}$$

by considering either the contexts $\mathbf{C}\{\} \equiv \{\}((\lambda y:1.y) (\lambda y:2.y))$ and $\mathbf{E}\{\} \equiv \{\} \Delta_{11} \wedge \{\} \Delta'_{11}$ or $\mathbf{C}\{\} \equiv ((\lambda y:1.y) (\lambda y:2.y))\{\}$ and $\mathbf{E}\{\} \equiv \Delta_{10} \{\} \wedge \Delta'_{10} \{\}$.

By making the first choice, we obtain:

$$(\lambda y:2.y) ((\lambda y:1.y) (\lambda y:2.y)) @ (\Delta_{13} \Delta_{11}) \wedge (\Delta'_{13} \Delta'_{11})$$

where the two overlapping contexts are: $\mathbf{C}\{\} \equiv (\lambda y:2.y) \{\}$ and $\mathbf{E}\{\} \equiv (\Delta_{13} \{\}) \wedge (\Delta'_{13} \{\})$.

Reducing this β -redex leads to the following term

$$(\lambda y:2.y) (\lambda y:2.y) @ (\Delta_{13} \Delta_{15}) \wedge (\Delta'_{13} \Delta'_{15})$$

where the overlapping contexts $\mathbf{C}\{\} \equiv \{\}$ and $\mathbf{E}\{\} \equiv \{\} \wedge \{\}$ individuate the last β -redex. Reducing this β -redex we obtain the term in normal form

$$(\lambda y:2.y) @ \Delta_{15} \wedge \Delta'_{15}$$

5. The isomorphism between Λ_{\wedge}^u and Λ_{\wedge}^t

In this section, we prove that the type system \vdash for Λ_{\wedge}^t is isomorphic to the classical system \vdash_{\wedge} for Λ_{\wedge}^u of Coppo and Dezani [6]. The isomorphism is given for a customization of the general definition of isomorphism given in [8, 14, 4], to the case of intersection-types and proof-trees.

From the logical point of view, the existence of an isomorphism means that there is a one-to-one correspondence between the judgments that can be proved in the two systems, and the derivations correspond with each other rule by rule. In what follows, and with a little abuse of notation, marked-terms and untyped terms of the λ -calculus will be ranged over by M, N, \dots , the difference between marked-terms and untyped-terms being clear from the context (*i.e.* the judgment to be proved).

Definition 5.1 (*Church vs. Curry*).

(1) The type-erasing function $\mathcal{E} : \Lambda_{\wedge}^t \Rightarrow \Lambda$ is inductively defined on terms as follows:

$$\begin{aligned} \mathcal{E}(x@_) &\triangleq x \\ \mathcal{E}((\lambda x:\iota.M)@_) &\triangleq \lambda x.\mathcal{E}(M@_) \\ \mathcal{E}((M N)@_) &\triangleq \mathcal{E}(M@_) \mathcal{E}(N@_) \end{aligned}$$

\mathcal{E} can be extended to contexts in the following way:

$$\begin{aligned} \mathcal{E}(\epsilon) &\triangleq \epsilon \\ \mathcal{E}(\Gamma, x@:\iota:\sigma) &\triangleq \mathcal{E}(\Gamma), x:\sigma \end{aligned}$$

$$\begin{aligned}
\mathcal{G}\left(\frac{x:\sigma \in E}{E \vdash_{\wedge} x : \sigma}^{(Var)}\right) &\triangleq \begin{cases} \frac{x@l:\sigma \in \Gamma}{\Gamma \vdash x@l : \sigma}^{(Var)} \\ \mathcal{E}(\Gamma) = E \quad l \text{ is fresh} \end{cases} \\
\mathcal{G}\left(\frac{\mathcal{D} : E, x:\sigma_1 \vdash_{\wedge} M' : \sigma_2}{E \vdash_{\wedge} \lambda x.M' : \sigma_1 \rightarrow \sigma_2}^{(\rightarrow I)}\right) &\triangleq \begin{cases} \frac{\mathcal{G}(\mathcal{D}) : \Gamma, x@l:\sigma_1 \vdash M@ \Delta : \sigma_2}{\Gamma \vdash (\lambda x:l.M)@(\lambda l:\sigma_1.\Delta) : \sigma_1 \rightarrow \sigma_2}^{(\rightarrow I)} \\ \mathcal{E}(\Gamma, x@l:\sigma_1) = E, x:\sigma_1 \ \& \ \mathcal{E}(M@ \Delta) = M' \end{cases} \\
\mathcal{G}\left(\frac{\begin{array}{c} \mathcal{D}_1 : E \vdash_{\wedge} M' : \sigma_1 \rightarrow \sigma_2 \\ \mathcal{D}_2 : E \vdash_{\wedge} N' : \sigma_1 \end{array}}{E \vdash_{\wedge} M' N' : \sigma_2}^{(\rightarrow E)}\right) &\triangleq \begin{cases} \frac{\mathcal{G}(\mathcal{D}_1) : \Gamma \vdash M@ \Delta_1 : \sigma_1 \rightarrow \sigma_2 \quad \mathcal{G}(\mathcal{D}_2) : \Gamma \vdash N@ \Delta_2 : \sigma_1}{\Gamma \vdash (M N)@(\Delta_1 \Delta_2) : \sigma_2}^{(\rightarrow E)} \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@ \Delta_1) = M' \ \& \ \mathcal{E}(N@ \Delta_2) = N' \end{cases} \\
\mathcal{G}\left(\frac{\begin{array}{c} \mathcal{D}_1 : E \vdash_{\wedge} M' : \sigma_1 \\ \mathcal{D}_2 : E \vdash_{\wedge} M' : \sigma_2 \end{array}}{E \vdash_{\wedge} M' : \sigma_1 \wedge \sigma_2}^{(\wedge I)}\right) &\triangleq \begin{cases} \frac{\mathcal{G}(\mathcal{D}_1) : \Gamma \vdash M@ \Delta_1 : \sigma_1 \quad \mathcal{G}(\mathcal{D}_2) : \Gamma \vdash M@ \Delta_2 : \sigma_2}{\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2}^{(\wedge I)} \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@(\Delta_1 \wedge \Delta_2)) = M' \end{cases} \\
\mathcal{G}\left(\frac{\mathcal{D} : E \vdash_{\wedge} M' : \sigma_1 \wedge \sigma_2}{E \vdash_{\wedge} M' : \sigma_1}^{(\wedge E_L)}\right) &\triangleq \begin{cases} \frac{\mathcal{G}(\mathcal{D}) : \Gamma \vdash M@ \Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\downarrow \Delta) : \sigma_1}^{(\wedge E_L)} \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@ \Delta) = M' \end{cases} \\
\mathcal{G}\left(\frac{\mathcal{D} : E \vdash_{\wedge} M' : \sigma_1 \wedge \sigma_2}{E \vdash_{\wedge} M' : \sigma_2}^{(\wedge E_R)}\right) &\triangleq \begin{cases} \frac{\mathcal{G}(\mathcal{D}) : \Gamma \vdash M@ \Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\Delta_{\downarrow}) : \sigma_2}^{(\wedge E_R)} \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@ \Delta) = M' \end{cases}
\end{aligned}$$

Fig. 5. The function \mathcal{G} .

- (2) Let $Der\Lambda_{\wedge}^u$ and $Der\Lambda_{\wedge}^t$ be the sets of all (un)typed derivations in \vdash_{\wedge} and \vdash, v , respectively. Let $\mathcal{D}, \mathcal{D}_1, \dots, \mathcal{D}_n$ range over (un)typed derivations. Systems \vdash_{\wedge} and \vdash are *isomorphic*, if and only if there exist $\mathcal{F} : Der\Lambda_{\wedge}^t \Rightarrow Der\Lambda_{\wedge}^u$ and $\mathcal{G} : Der\Lambda_{\wedge}^u \Rightarrow Der\Lambda_{\wedge}^t$, such that:
- (a) (Soundness) If $\mathcal{D} : \Gamma \vdash M@ \Delta : \sigma$, then $\mathcal{F}(\mathcal{D}) : \mathcal{E}(\Gamma) \vdash_{\wedge} \mathcal{E}(M@ \Delta) : \sigma$;
 - (b) (Completeness) If $\mathcal{D} : E \vdash_{\wedge} M' : \sigma$, then there exists Γ and Δ , such that $\mathcal{G}(\mathcal{D}) : \Gamma \vdash M@ \Delta : \sigma$, and $\mathcal{E}(\Gamma) \equiv E$, with $\mathcal{E}(M@ \Delta) \equiv M'$;
 - (c) (Inversion) $\mathcal{F} \circ \mathcal{G}$ is the identity in $Der\Lambda_{\wedge}^u$, and $\mathcal{G} \circ \mathcal{F}$ is the identity in $Der\Lambda_{\wedge}^t$, modulo uniform naming of variable-marks. That is,

$$\mathcal{G}(\mathcal{F}(\Gamma \vdash M@ \Delta : \sigma)) = \text{ren}(\Gamma) \vdash \text{ren}(M@ \Delta) : \sigma$$

where ren is a simple function renaming the free occurrences of variable-marks;

1. (Faithfulness) Both \mathcal{F} and \mathcal{G} preserve the structure of derivations, (*i.e.*, the tree obtained from a derivation by erasing all judgments, but not the names of the rules).

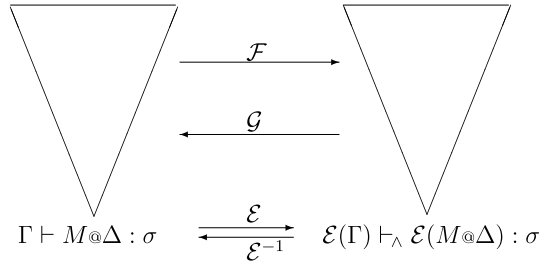
Function \mathcal{F} and \mathcal{G} are described in Figs. 6 and 5.

Notice that the definition of isomorphism expresses more than just soundness and completeness of \mathcal{E} . Indeed, soundness and completeness imply an isomorphism between the judgments of the two systems, but they do not imply necessarily a one-one correspondence between derivations. Fig. 7 shows the various functions between typed and untyped systems of λ -calculi that realize the above relations between typed and untyped judgments and derivations.

Theorem 5.1 (Isomorphism). *The systems \vdash and \vdash_{\wedge} are isomorphic.*

Proof. Soundness can be proved by induction on the structure of the derivation in the Λ_{\wedge}^t . Completeness can be

$$\begin{aligned}
\mathcal{F}\left(\frac{x@l:\sigma \in \Gamma}{\Gamma \vdash x@l:\sigma} (Var)\right) &\triangleq \begin{cases} \frac{x:\sigma \in E}{E \vdash_\wedge x:\sigma} (Var) \\ \mathcal{E}(\Gamma) = E \end{cases} \\
\mathcal{F}\left(\frac{\Gamma, x@l:\sigma_1 \vdash M@ \Delta : \sigma_2}{\Gamma \vdash (\lambda x:l.M)@(\lambda l:\sigma_1.\Delta) : \sigma_1 \rightarrow \sigma_2} (\rightarrow I)\right) &\triangleq \begin{cases} \frac{\mathcal{F}(\mathcal{D}) : E, x:\sigma_1 \vdash_\wedge M' : \sigma_2}{E \vdash_\wedge \lambda x.M' : \sigma_1 \rightarrow \sigma_2} (\rightarrow I) \\ \mathcal{E}(\Gamma, x@l:\sigma_1) = E, x:\sigma_1 \ \& \ \mathcal{E}(M@ \Delta) = M' \end{cases} \\
\mathcal{F}\left(\frac{\mathcal{D}_1 : \Gamma \vdash M@ \Delta_1 : \sigma_1 \rightarrow \sigma_2 \quad \mathcal{D}_2 : \Gamma \vdash N@ \Delta_2 : \sigma_1}{\Gamma \vdash (MN)@(\Delta_1 \Delta_2) : \sigma_2} (\rightarrow E)\right) &\triangleq \begin{cases} \frac{\mathcal{F}(\mathcal{D}_1) : E \vdash_\wedge M' : \sigma_1 \rightarrow \sigma_2 \quad \mathcal{F}(\mathcal{D}_2) : E \vdash_\wedge N' : \sigma_1}{E \vdash_\wedge M' N' : \sigma_2} (\rightarrow E) \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@ \Delta_1) = M' \ \& \ \mathcal{E}(N@ \Delta_2) = N' \end{cases} \\
\mathcal{F}\left(\frac{\mathcal{D}_1 : \Gamma \vdash M@ \Delta_1 : \sigma_1 \quad \mathcal{D}_2 : \Gamma \vdash M@ \Delta_2 : \sigma_2}{\Gamma \vdash M@(\Delta_1 \wedge \Delta_2) : \sigma_1 \wedge \sigma_2} (\wedge I)\right) &\triangleq \begin{cases} \frac{\mathcal{F}(\mathcal{D}_1) : E \vdash_\wedge M' : \sigma_1 \quad \mathcal{F}(\mathcal{D}_2) : E \vdash_\wedge M' : \sigma_2}{E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2} (\wedge I) \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@(\Delta_1 \wedge \Delta_2)) = M' \end{cases} \\
\mathcal{F}\left(\frac{\mathcal{D} : \Gamma \vdash M@ \Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\surd \Delta) : \sigma_1} (\wedge E_L)\right) &\triangleq \begin{cases} \frac{\mathcal{F}(\mathcal{D}) : E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}{E \vdash_\wedge M' : \sigma_1} (\wedge E_L) \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@ \Delta) = M' \end{cases} \\
\mathcal{F}\left(\frac{\mathcal{D} : \Gamma \vdash M@ \Delta : \sigma_1 \wedge \sigma_2}{\Gamma \vdash M@(\Delta \surd) : \sigma_2} (\wedge E_R)\right) &\triangleq \begin{cases} \frac{\mathcal{F}(\mathcal{D}) : E \vdash_\wedge M' : \sigma_1 \wedge \sigma_2}{E \vdash_\wedge M' : \sigma_2} (\wedge E_R) \\ \mathcal{E}(\Gamma) = E \ \& \ \mathcal{E}(M@ \Delta) = M' \end{cases}
\end{aligned}$$

Fig. 6. The function \mathcal{F} .Fig. 7. Functions between Λ_\wedge^t s and Λ_\wedge^u s Judgments and Derivations.

proved by induction on the structure of the derivation in Λ_\wedge^u , using soundness. Inversion can be proved by induction on the structure of both the derivations, using the soundness and completeness result. Faithfulness is immediate. \square

We can also explore the relationship between Λ_\wedge^t and the proof calculus $\Lambda\mathcal{P}$ by defining an erasure function $\mathcal{E}' : \Lambda_\wedge^t \Rightarrow \Lambda\mathcal{P}$ as follows:

$$\mathcal{E}'(_@ \Delta) \triangleq \Delta$$

The function \mathcal{E}' can be extended naturally to a function from contexts to proof-contexts:

$$\begin{aligned}
\mathcal{E}'(\epsilon) &\triangleq \epsilon \\
\mathcal{E}'(\Gamma, x@l:\sigma) &\triangleq \mathcal{E}'(\Gamma), l:\sigma
\end{aligned}$$

Then it is easy to define a function $\mathcal{F}' : \text{Der } \Lambda_{\wedge}^t \Rightarrow \text{Der } \Lambda\mathcal{P}$ such that $\mathcal{D} : \Gamma \vdash M @ \Delta : \sigma$ implies $\mathcal{F}'(\mathcal{D}) : \mathcal{E}'(\Gamma) \vdash_{\mathcal{P}} \Delta : \sigma$. \mathcal{F}' consists in just applying \mathcal{E}' to all contexts and subjects of the derivation. Note that Λ_{\wedge}^t and $\Lambda\mathcal{P}$ are not isomorphic; for example, the statement $\iota_1 : \sigma \vdash_{\mathcal{P}} (\lambda \iota_2 : \tau. \iota_2) \wedge \iota_1 : (\tau \rightarrow \tau) \wedge \sigma$ in $\Lambda\mathcal{P}$ has no a corresponding counterpart in Λ_{\wedge}^t .

6. Metatheory of Λ_{\wedge}^t

In this section, we will prove properties [3–6] listed in the introduction. First, we need to prove the Generation and Substitution lemmas.

Lemma 6.1 (Generation).

- (1) If $\Gamma \vdash M @ \Delta_1 \wedge \Delta_2 : \sigma$, then $\sigma \equiv \sigma_1 \wedge \sigma_2$ and $\Gamma \vdash M @ \Delta_1 : \sigma_1$ and $\Gamma \vdash M @ \Delta_2 : \sigma_2$;
- (2) If $\Gamma \vdash M @ \Delta : \sigma$ then there exists τ , such that $\Gamma \vdash M @ \Delta : \sigma \wedge \tau$;
- (3) If $\Gamma \vdash M @ \Delta_{\searrow} : \sigma$ then there exists τ , such that $\Gamma \vdash M @ \Delta : \tau \wedge \sigma$;
- (4) If $\Gamma \vdash x @ \iota : \sigma$, then $x @ \iota : \sigma \in \Gamma$;
- (5) If $\Gamma \vdash (\lambda x : \iota. M) @ (\lambda \iota : \sigma_1. \Delta) : \sigma_1 \rightarrow \sigma_2$, then $\Gamma, x @ \iota : \sigma_1 \vdash M @ \Delta : \sigma_2$;
- (6) If $\Gamma \vdash M N @ \Delta_1 \Delta_2 : \sigma$, then there exists σ' , such that $\Gamma \vdash M @ \Delta_1 : \sigma' \rightarrow \sigma$, and $\Gamma \vdash N @ \Delta_2 : \sigma'$.

Proof. By induction on derivation. \square

Lemma 6.2 (Substitution).

If $\Gamma, x @ \iota : \sigma \vdash M @ \Delta : \tau$, and $\Gamma \vdash N @ \Delta' : \sigma$, then $\Gamma \vdash M[N/x] @ \Delta[\Delta'/\iota] : \tau$.

Proof. By induction on derivation. \square

We continue with subject reduction

Theorem 6.1 (Subject reduction).

If $\Gamma \vdash M @ \Delta : \sigma$ and $M @ \Delta \rightarrow N @ \Delta'$, then $\Gamma \vdash N @ \Delta' : \sigma$.

Proof. We will treat the case \rightarrow is \rightarrow_{β} ; the cases \rightarrow_{π_1} and \rightarrow_{π_2} are easier. By definition, there are two contexts $\mathbf{C}\{\}$ and $\mathbf{E}\{\}$, such that $\mathbf{C}\{\} \sqsubseteq \mathbf{E}\{\}$, and $M @ \Delta \equiv \mathbf{C}\{(\lambda x : \iota. P) Q\} @ \mathbf{E}\{(\lambda \iota : \sigma_i. \Delta_i) \Delta'_i\}_{i \in I}$ which β -reduces to $\mathbf{C}\{P[Q/x]\} @ \mathbf{E}\{\Delta_i[\Delta'_i/\iota]\}_{i \in I}$.

By induction on $\mathbf{C}\{\}$. Let $\mathbf{C}\{\} \equiv \{\}$. We proceed now by induction on $\mathbf{E}\{\}$. If $\mathbf{E}\{\} \equiv \{\}$, then, by Generation Lemma.(5 – 6) the derivation is of the shape:

$$\frac{\frac{\Gamma, x @ \iota : \tau \vdash P @ \Delta : \sigma}{\Gamma \vdash \lambda x : \iota. P @ \lambda \iota : \tau. \Delta : \tau \rightarrow \sigma} (\rightarrow I) \quad \Gamma \vdash Q @ \Delta' : \tau}{\Gamma \vdash (\lambda x : \iota. P) Q @ (\lambda \iota : \tau. \Delta) \Delta' : \sigma} (\rightarrow E)$$

Then, by Substitution Lemma, we get $\Gamma \vdash P[Q/x] @ \Delta[\Delta'/\iota] : \sigma$.

Let $\mathbf{E}\{\} \equiv \mathbf{E}_1\{\} \wedge \mathbf{E}_2\{\}$, where $\{\} \sqsubseteq \mathbf{E}_i\{\}$, and $1 \leq i \leq 2$. Then, by Generation Lemma.(1), we have $\sigma \equiv \sigma_1 \wedge \sigma_2$ and $\Gamma \vdash (\lambda x : \iota. P) Q @ \mathbf{E}_1\{(\lambda \iota : \sigma_i. \Delta_i) \Delta'_i\}_{i \in I_1} : \sigma_1$ and $\Gamma \vdash (\lambda x : \iota. P) Q @ \mathbf{E}_2\{(\lambda \iota : \sigma_i. \Delta_i) \Delta'_i\}_{i \in I_2} : \sigma_2$, for some I_1, I_2 , such that $I = I_1 \cup I_2$. By induction, we have $\Gamma \vdash P[Q/x] @ \mathbf{E}_1\{\Delta_i[\Delta'_i/\iota]\}_{i \in I_1} : \sigma_1$, and $\Gamma \vdash P[Q/x] @ \mathbf{E}_2\{\Delta_i[\Delta'_i/\iota]\}_{i \in I_2} : \sigma_2$, and the proof follows, by rule $(\wedge I)$.

The cases for $\mathbf{E}\{\} \equiv \searrow \mathbf{E}_1\{\}$, or $\mathbf{E}\{\} \equiv \mathbf{E}_1\{\} \searrow$ follow easily by induction. No other cases can apply.

For the inductive case, we will show the proof in case $\mathbf{C}\{\} \equiv R \mathbf{C}_1\{\}$, and $\mathbf{E}\{\} \equiv \Delta \mathbf{E}_1\{\}$, with $\mathbf{C}_1\{\} \sqsubseteq \mathbf{E}_1\{\}$. Then $R \mathbf{C}_1\{(\lambda x : \iota. P) Q\} @ \Delta \mathbf{E}_1\{(\lambda \iota : \sigma_i. \Delta_i) \Delta'_i\}_{i \in I} \rightarrow_{\beta} R \mathbf{C}_1\{P[Q/x]\} @ \Delta \mathbf{E}_1\{\Delta_i[\Delta'_i/\iota]\}_{i \in I}$. By the Generation Lemma, there exists σ' , such that $\Gamma \vdash R @ \Delta : \sigma' \rightarrow \sigma$, and $\Gamma \vdash \mathbf{C}_1\{(\lambda x : \iota. P) Q\} @ \mathbf{E}_1\{(\lambda \iota : \sigma_i. \Delta_i) \Delta'_i\}_{i \in I} : \sigma'$. By induction we get $\Gamma \vdash \mathbf{C}_1\{P[Q/x]\} @ \mathbf{E}_1\{\Delta_i[\Delta'_i/\iota]\}_{i \in I} : \sigma'$, and the proof follows by rule $(\rightarrow E)$. The other cases are easy. \square

The strong normalization of Λ_{\wedge}^t is proved from the strong normalization of $\Lambda\mathcal{P}$.

Theorem 6.2 (Strong normalization of Λ_{\wedge}^t).

Λ_{\wedge}^t is strongly normalizing.

Proof. Let $\mathcal{D} : \Gamma \vdash M @ \Delta : \sigma$, and let us assume, by *absurdum*, that there is an infinite reduction sequence starting from $M @ \Delta$, i.e.,

$$M @ \Delta \equiv C_1\{M_1\} @ A_1\{\Delta_1^i\}_{i \in I_1} \rightarrow C_2\{M_2\} @ A_2\{\Delta_2^i\}_{i \in I_2} \rightarrow \dots \rightarrow \\ C_j\{M_j\} @ A_j\{\Delta_j^i\}_{i \in I_j} \rightarrow C_{j+1}\{M_{j+1}\} @ A_{j+1}\{\Delta_{j+1}^i\}_{i \in I_{j+1}} \rightarrow \dots$$

where $A\{\}$ is a context either of the shape $E\{\}$, or of the shape $D\{\}$. In the first case the reduction is a β -reduction, in the second one is a π -reduction. By applying the function \mathcal{F}' , defined at the end of Section 5, on each element of the reduction sequence, we get an infinite sequence of proof-terms $A_1\{\Delta_1^i\}_{i \in I_1}, A_2\{\Delta_2^i\}_{i \in I_2} \dots A_j\{\Delta_j^i\}_{i \in I_j}, A_{j+1}\{\Delta_{j+1}^i\}_{i \in I_{j+1}} \dots$, such that, for all $j \geq 0$, we have that $A_j\{\Delta_j^i\}_{i \in I_j}$ reduces to $A_{j+1}\{\Delta_{j+1}^i\}_{i \in I_{j+1}}$, either by a π -reduction or by a strictly positive number of β -reductions. But this is impossible, since Λ^t enjoys the strong normalization property (Fact 1). \square

The further requirement we asked for is the unicity of typing. In general, for typed languages, typing is unique modulo α -conversion, i.e., modulo renaming of bound-variables.

Theorem 6.3 (Unicity of typing of Λ^t).

If $\mathcal{D}_1 : \Gamma \vdash M @ \Delta : \sigma_1$, and $\mathcal{D}_2 : \Gamma \vdash M' @ \Delta' : \sigma_2$, and $M @ \Delta =_\alpha M' @ \Delta'$, then $\sigma_1 \equiv \sigma_2$, and $\mathcal{D}_1 \equiv \mathcal{D}_2$.

Proof. By easy induction on the structure of the derivation \mathcal{D}_1 . \square

We can finish this section by presenting the type reconstruction and the type checking algorithms for Λ^t in Fig. 8, and by proving that they are decidable. The soundness and completeness proofs follow.

Theorem 6.4 (Type reconstruction for Λ^t).

(Soundness) If $\text{Type}_\wedge(\Gamma, M @ \Delta) = \sigma$, then $\Gamma \vdash M @ \Delta : \sigma$

(Completeness) If $\Gamma \vdash M @ \Delta : \sigma$, then $\text{Type}_\wedge(\Gamma, M @ \Delta) = \sigma$.

$\text{Type}_\wedge(\Gamma, M @ \Delta)$	\triangleq	match $(M @ \Delta)$ with
$(_ @ (\Delta_1))$	\Rightarrow^1	σ_1 if $\text{Type}_\wedge(\Gamma, M @ \Delta_1) = \sigma_1 \wedge \sigma_2$
$(_ @ (\Delta_1 \downarrow))$	\Rightarrow^2	σ_2 if $\text{Type}_\wedge(\Gamma, M @ \Delta_1) = \sigma_1 \wedge \sigma_2$
$(_ @ (\Delta_1 \wedge \Delta_2))$	\Rightarrow^3	$\sigma_1 \wedge \sigma_2$ if $\text{Type}_\wedge(\Gamma, M @ \Delta_1) = \sigma_1$ and $\text{Type}_\wedge(\Gamma, M @ \Delta_2) = \sigma_2$
$(x @ _)$	\Rightarrow^4	σ if $x @ \iota : \sigma \in \Gamma$
$((\lambda x : \iota. M_1) @ (\lambda \iota : \sigma_1. \Delta_1))$	\Rightarrow^5	$\sigma_1 \rightarrow \sigma_2$ if $\text{Type}_\wedge((\Gamma, x @ \iota : \sigma_1), M_1 @ \Delta_1) = \sigma_2$
$((M_1 M_2) @ (\Delta_1 \Delta_2))$	\Rightarrow^6	σ_2 if $\text{Type}_\wedge(\Gamma, M_1 @ \Delta_1) = \sigma_1 \rightarrow \sigma_2$ and $\text{Type}_\wedge(\Gamma, M_2 @ \Delta_2) = \sigma_1$
$(_ @ _)$	\Rightarrow^7	false otherwise

$$\text{Typecheck}_\wedge(\Gamma, M @ \Delta, \sigma) \triangleq \text{Type}_\wedge(\Gamma, M @ \Delta) \stackrel{?}{=} \sigma$$

Fig. 8. The type reconstruction and type checking Algorithms for Λ^t .

Proof.

(Soundness) By induction on the structure of $(M@Δ)$.

- $(_@(\swarrow Δ_1))$ Then $Δ ≡ \swarrow Δ_1$ and $σ ≡ σ_1$. By induction, the judgment $Γ ⊢ M@Δ_1 : σ_1 ∧ σ_2$ is derivable. Apply rule $(\wedge E_L)$ to obtain a derivation for $Γ ⊢ M@(\swarrow Δ_1) : σ_1$.
- $(_@(\Delta_1 \searrow))$ Then $Δ ≡ \Delta_1 \searrow$ and $σ ≡ σ_2$. By induction, the judgment $Γ ⊢ M@Δ_1 : σ_1 ∧ σ_2$ is derivable. Apply rule $(\wedge E_R)$ to obtain a derivation for $Γ ⊢ M@(\Delta_1 \searrow) : σ_2$.
- $(_@(\Delta_1 \wedge \Delta_2))$ Then $σ ≡ σ_1 ∧ σ_2$. By induction, the judgments $Γ ⊢ M@Δ_1 : σ_1$ and $Γ ⊢ M@Δ_2 : σ_2$ are derivable. Apply rule $(\wedge I)$ to obtain a derivation for $Γ ⊢ M@(\Delta_1 \wedge \Delta_2) : σ_1 ∧ σ_2$.
- $(x@_)$ Then $M ≡ x$ and $Δ ≡ \iota$, since the Type_\wedge algorithm (that works via a classical ML-like match-case analysis) has already ruled out the cases of $Δ \in \{\swarrow \Delta_1, \Delta_1 \searrow, \Delta_1 \wedge \Delta_2\}$, and since the case $Δ ≡ \lambda \iota : \sigma_1. \Delta_1$ does not apply. By hypothesis we get $x@ \iota : \sigma \in \Gamma$. Apply rule (Var) to obtain a derivation for $Γ ⊢ x@ \iota : \sigma$.
- $((\lambda x : \iota. M_1)@(\lambda \iota : \sigma_1. \Delta_1))$ Then $M ≡ \lambda x : \iota. M_1$ and $Δ ≡ \lambda \iota : \sigma_1. \Delta_1$ and $σ ≡ σ_1 \rightarrow σ_2$. By induction, the judgment $Γ, x@ \iota : \sigma_1 ⊢ M_1@\Delta_1 : σ_2$ is derivable. Apply rule $(\rightarrow I)$ to obtain a derivation for $Γ ⊢ (\lambda x : \iota. M_1)@(\lambda \iota : \sigma_1. \Delta_1) : σ_1 \rightarrow σ_2$.
- $((M_1 M_2)@(\Delta_1 \Delta_2))$ Then $M ≡ M_1 M_2$ and $Δ ≡ \Delta_1 \Delta_2$ and $σ ≡ σ_2$. By induction, the judgments $Γ ⊢ M_1@\Delta_1 : σ_1 \rightarrow σ_2$ and $Γ ⊢ M_2@\Delta_2 : σ_1$ are derivable. Apply rule $(\rightarrow E)$ to obtain a derivation for $Γ ⊢ (M_1 M_2)@(\Delta_1 \Delta_2) : σ_2$.
- $(_@_)$ This case does not apply since $σ \neq \text{false}$.

(Completeness) By induction on the derivation of $Γ ⊢ M@Δ : σ$.

- (Var) Then $M ≡ x$ and $Δ ≡ \iota$. By match-case number 4 we get $\text{Type}_\wedge(x, \iota@σ) = σ$.
- $(\rightarrow I)$ Then $M ≡ \lambda x : \iota. M_1$ and $Δ ≡ \lambda \iota : \sigma_1. \Delta_1$ and $σ ≡ σ_1 \rightarrow σ_2$. By induction we get $\text{Type}_\wedge((\Gamma, x@ \iota : \sigma_1), M_1@\Delta_1) = σ_2$, and by match-case 5 we get $\text{Type}_\wedge((\lambda x : \iota. M_1)@(\lambda \iota : \sigma_1. \Delta_1)) = σ_1 \rightarrow σ_2$.
- $(\rightarrow E)$ Then $M ≡ M_1 M_2$ and $Δ ≡ \Delta_1 \Delta_2$ and $σ ≡ σ_2$. By induction we get $\text{Type}_\wedge(\Gamma, M_1@\Delta_1) = σ_1 \rightarrow σ_2$ and $\text{Type}_\wedge(\Gamma, M_2@\Delta_2) = σ_1$, and by match-case 6 we get $\text{Type}_\wedge((M_1 M_2)@(\Delta_1 \Delta_2)) = σ_2$.
- $(\wedge I)$ Then $Δ ≡ \Delta_1 \wedge \Delta_2$ and $σ ≡ σ_1 ∧ σ_2$. By induction we get $\text{Type}_\wedge(M, \Delta_1) = σ_1$ and $\text{Type}_\wedge(M, \Delta_2) = σ_2$, and by match-case 3 we get $\text{Type}_\wedge(M, (\Delta_1 \wedge \Delta_2)) = σ_1 ∧ σ_2$.
- $(\wedge E_L)$ Then $Δ ≡ \swarrow \Delta_1$ and $σ ≡ σ_1$. By induction we get $\text{Type}_\wedge(\Gamma, M@\Delta_1) = σ_1 ∧ σ_2$, and by match-case 1 we get $\text{Type}_\wedge(M, \swarrow \Delta_1) = σ_1$.
- $(\wedge E_R)$ Then $Δ ≡ \Delta_1 \searrow$ and $σ ≡ σ_2$. By induction we get $\text{Type}_\wedge(\Gamma, M@\Delta_1) = σ_1 ∧ σ_2$, and by match-case 2 we get $\text{Type}_\wedge(M, \Delta_1 \searrow) = σ_2$. \square

Theorem 6.5 (Type checking for Λ_\wedge^t).

$Γ ⊢ M@Δ : σ$, if and only if $\text{Typecheck}_\wedge(\Gamma, M@Δ, \sigma) = \text{true}$.

Proof. The \Rightarrow part can be proved using completeness of the type reconstruction algorithm (Theorem 6.4), while the \Leftarrow part can be proved using soundness of the type reconstruction algorithm. \square

Theorem 6.6 (Judgment decidability).

If is decidable whether the Λ_\wedge^t judgment $Γ ⊢ M@Δ : σ$ is derivable.

Proof. Routine. \square

7. Conclusions

We studied in this paper the problem of designing a λ -calculus *à la* Church corresponding to the intersection-type assignment system. In particular, we asked for a typed language such that its relationship with the intersection-type assignment system enjoys all the standard requirements we posed in [8, 14, 4]. Examples of such “good” correspondences are respectively, the Church and Curry version of the simple typed λ -calculus (if written using the same symbols), and the typed and type assignment version of the second order λ -calculus [9, 13]. We succeed in designing a calculus based essentially on two basic and simple ideas: an imperative-like notion of typing, when types are assigned to variables “at a given mark”, and a proof-calculus, describing intersection-type derivations, whose terms are used as proof-trees for the terms of the target calculus.

A reader interested in particular in programming applications could object that the used language is far from being “usable”, since the user needs to specify not only the typed terms, but also their proof-trees, which are encoding of type-derivations. The answer can be twofold. From a programming languages point of view, in every typed language the user, in order to write explicitly the type of a term, in some sense needs to “guess” the correct type-derivation assigning that type to the term itself. Here, obviously the type-derivations are more difficult than in the simple typed case. But if we think, for example, to Girard’s Second Order Typed λ -calculus [9], in order to write the term

$$\Lambda\beta.\Lambda\gamma.\lambda x:(\forall\alpha.\alpha).x(\beta\rightarrow\gamma) \text{ of type } \forall\beta.\forall\gamma.(\forall\alpha.\alpha)\rightarrow(\beta\rightarrow\gamma)$$

one needs to know exactly how the rules for introducing and eliminating the universal quantifier work. However, we think that the production of an usable language is not the only justification for the problem we studied, as it was especially for [17, 16]. The relationship between typed and type assignment systems is an important theoretical issue, that is interesting in itself.

Acknowledgments

Simona was kindly supported by *QSL: Qualité et Sécurité du Logiciel*, CPER, Région Lorraine, Nancy, and by INRIA; Luigi was supported by the French CNRS grant *ACI Modulogic*. Moreover, both authors want warmly thank the two anonymous referees, for their very sharp and constructive suggestions.

References

- [2] H. Barendregt, M. Coppo, M. Dezani-Ciancaglini, A filter lambda model and the completeness of type assignment, *Journal of Symbolic Logic* 48 (4) (1983) 931–940.
- [3] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, G. Longo, Extended type structures and filter lambda models, *Logic Colloquium '82*, North Holland, 1983, pp. 241–262.
- [4] S. vanBakel, L. Liquori, S. Ronchi Della Rocca, P. Urzyczyn, Comparing cubes of typed and type assignment systems, *Annals of Pure and Applied Logic* 86 (3) (1997) 267–303.
- [5] B. Capitani, B. Venneri, Hyperformulae, parallel deductions and intersection-types, in: *Proceedings of the BOTH 2001, ENTCS*, 50(2), 2001, pp. 180–198.
- [6] M. Coppo, M. Dezani-Ciancaglini, An extension of the basic functionality theory for the λ -calculus, *Notre Dame Journal of Formal Logic* 21 (4) (1980) 685–693.
- [7] M. Dezani-Ciancaglini, E. Giovannetti, U. de’iguoro, Intersection-types, Lambda-models and Böhm Trees, in: *MSJ-Memoir*, vol. 2, *Theories of Types and Proofs*, vol. 2, Mathematical Society of Japan, 1998, pp. 45–97.
- [8] P. Giannini, F. Honsell, S. Ronchi Della Rocca, Type inference: some results, some problems, *Fundamenta Informaticae* 19 (1,2) (1993) 87–126.
- [9] J.Y. Girard, The system F of variable types, fifteen years later, *Theoretical Computer Science* 45 (1986) 159–192.
- [10] J.R. Hindley, Coppo Dezani types do not correspond to propositional logic, *Theoretical Computer Science* 28 (1–2) (1984) 235–236.
- [11] W.A. Howard, The formulae-as-types notion of construction, in: H.B. Curry (Ed.), *Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, London, 1980, pp. 479–490.
- [12] J.L. Krivine, *Lambda-calcul, Types et Modèles*, Masson, 1990.
- [13] D. Leivant, Polymorphic type inference, in: *Proceedings of POPL*, ACM Press, 1983.
- [14] L. Liquori, Type assignment systems for lambda calculi and for the lambda calculus of objects, Ph.D. Thesis, University of Turin, 1996, 193 pp.
- [15] G. Pottinger, A type assignment for the strongly normalizable λ -terms, in: H.B. Curry (Ed.), *Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, London, 1980, pp. 561–577.
- [16] B.C. Pierce, D.N. Turner, Simple type-theoretic foundations for object-oriented programming, *Journal of Functional Programming* 4 (2) (1994) 207–247.
- [17] J.C. Reynolds, Design of the Programming Language Forsythe, in: O’Hearn and Tennent (Eds.), *Algol-like Languages*, Birkhauser, 1996.
- [18] S. Ronchi Della Rocca, Intersection-typed lambda-calculus, in: *Proceedings of ICTRS, ENTCS*, 70(1), 2002.
- [19] S. Ronchi Della Rocca, L. Roversi, Intersection logic, in: *Proceedings of CSL, LNCS* 2142, Springer-Verlag, 2001, pp. 414–428.
- [20] J.B. Wells, A. Dimock, R. Muller, F. Turbak, A calculus with polymorphic and polyvariant flow types, *Journal of Functional Programming* 12 (3) (2002) 183–227.
- [21] J.B. Wells, C. Haack, Branching types, in: *Programming Languages & Systems*, 11th European Symposium Programming, LNCS 2305, Springer-Verlag, 2002, pp. 115–132.